The opinion in support of the decision being entered today was not written for publication and is not binding precedent of the Board.

Paper No. 18

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

*Ex parte* VITA BORTNIKOV,
DAVID JOHN LAMBERT,
BILHA MENDELSON,
ROBERT RALPH ROEDIGER,
WILLIAM JON SCHMIDT,
and INBAL SHAVIT-LOTTEM

Appeal No. 2001-0653
Application 08/820,736[1]

ON BRIEF

Before JERRY SMITH, BARRETT, and SAADAT, *Administrative Patent Judges*.

BARRETT, *Administrative Patent Judge*.

## DECISION ON APPEAL

This is a decision on appeal under 35 U.S.C. § 134 from the final rejection of claims 1 and 3-40.

[1] Application for patent filed March 19, 1997, entitled "System and Method for Generating and Utilizing Organized Profile Information."

We affirm-in-part.

## BACKGROUND

The disclosed invention relates to an optimizing compiler that uses profile data. Whenever a modification is made to a procedure that causes a change to its flow control chart, the profile information previously gathered for the procedure will often be at least partially invalid or incomplete (spec. at 13). Prior art compilers are unable to use existing profile data in such cases, and must generate new profile data for the whole program each time a procedure is modified (spec. at 13). With large programs, the time and expense involved in re-profiling the program each time a minor bug fix occurs may be significant (spec. at 3-4). The present invention allows the use of existing profile information even if source code modifications have taken place by identifying invalid profile information and skipping profile data of only those procedures (spec. at 13-14).

Claim 1 is reproduced below.

1. A program product, said program product comprising:

storage media; and

an instrumented executable program module stored on said storage media, said module comprising a mechanism that causes profile information to be generated into at least one procedure specific data storage area each time an instrumented code block is executed.

The examiner relies on the following references:

Turbo Profiler Version 2.0 User's Guide (Borland International Inc. 1991) (hereinafter "Profiler").

Aho et al. (Aho), Compilers -- Principles, Techniques, and Tools (Addison-Wesley Pub. Co. 1986), Chaps. 7 & 10.

Claim 26 stands rejected under 35 U.S.C. § 112, second paragraph, as being indefinite.

Claims 1 and 3-40 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Profiler in view of common knowledge of compiler theory as taught by Aho.

We refer to the final rejection (Paper No. 11) (pages referred to as "FR__") and the examiner's answer (Paper No. 17) (pages referred to as "EA__") for a statement of the examiner's rejection, and to the appeal brief (Paper No. 16) (pages referred to as "Br__") for a statement of appellants' arguments thereagainst.

## OPINION

### Indefiniteness

In the second Office action (Paper No. 7), the examiner rejected claims 10 and 26 under 35 U.S.C. § 112, second paragraph, as indefinite. The examiner quoted claim 10, which read: "The apparatus of claim 8 wherein said signature of each procedure includes at least one functional value computed from attributes of said procedure." The examiner stated (Paper No. 7,

p. 2): "This claim was interpreted as input to the profiler which can/should not be contrrolled [sic] by a Profiler. Not a limitation of a Profiler." Claim 26 was not mentioned in the reasoning, but it contains similar language: "The program product of claim 23 wherein said mechanism that determines if procedure specific profile data is valid examines at least one functional value computed from the attributes of the related procedure." Presumably, the same reasoning was intended to apply to claim 26.

Claim 10 was amended by incorporating the limitations of claim 8 to read (Paper No. 10): "<u>The apparatus of claim 5 wherein said checking mechanism determines validity of each of said at least one procedure counter area by comparing a signature of each procedure with information stored in each corresponding procedure counter area</u> [The apparatus of claim 8] wherein said signature of each procedure includes at least one functional value computed from attributes of said procedure."

In the final rejection (Paper No. 11), the examiner withdrew the rejection of claim 10 but maintained the rejection of claim 26 stating "the language is vague and indefinite" (FR2).

Appellants argue that the examiner provided no explanation of the basis for the rejection, but merely states that the language is "vague and indefinite" (Br7).

The examiner responds that "[claims 10 and 26] were initially identical and both received the same rejection" (EA9) and repeats the rejection of claims 10 and 26 from Paper No. 7 and the final rejection, Paper No. 11 (EA9-10).

Because claim 26 is similar to original claim 10, we presume that the same reasoning was intended to apply, although we agree with appellants that the rejection is not express on this point. The examiner stated (Paper No. 7, p. 2): "This claim was interpreted as input to the profiler which can/should not be contrrolled [sic] by a Profiler. Not a limitation of a Profiler." We do not understand this reasoning and the rejection has not been further explained. It is not even clear whether the "Profiler" is meant to refer to the Profiler reference or to profilers in general. Nor do we understand why the examiner withdrew the rejection of claim 10 if he maintains the rejection of claim 26 since claim 10 was only amended to add limitations of original claim 8, from which it depended, and the original rejected language remains unchanged.

Nevertheless, we see nothing indefinite about claim 26. The specification states (spec. at 23):

> Determining whether or not a procedure has a valid PCA [procedure counter area] may be accomplished by comparing a "signature" of the procedure with information in the PCA. For example, the optimization mechanism 19 can compare the number of counters in the PCA with appropriate number of counters required in the procedure being processed. The optimization mechanism 19 could also compare a check sum in the PCA with a calculated check sum for the procedure.

It seems that "functional value computed from attributes of the related procedure" in claim 26 could read on the disclosed calculated check sum for the procedure. As another example, in programming it is common to use a "make file" which is a set of instructions (usually ASCII Text) to build a program. The make utility reads the dependencies, figures out which items need to be rebuilt (for example you changed the source code of a module after the last time it was built) and automates the process by executing the appropriate set of commands. The process of determining which modules were changed must use some attributes of the module and is one way of determining if specific profile data is valid. Thus, we conclude that the examiner has not established a <u>prima facie</u> case of indefiniteness. The rejection of claim 26 is reversed.

<u>Obviousness</u>

<u>Grouping of claims</u>

Appellants identify the following groups of claims, with the individual claims within each claim group standing or falling together (Br6). The representative claim in each group, as argued by appellants, is underlined; the representative claims chosen by the examiner (EA7) differ for Groups 4 and 5, but this does not affect the analysis.

Group 1 - Claims <u>1</u>, 3, 4, 20-22, 32, 34, and 40;

Group 2 - Claim 33;
Group 3 - Claims 5-7, 13-17, 23, 24, 29, 30, 35, 36, and 39;
Group 4 - Claims 8-10, 25, 26, and 37;
Group 5 - Claims 11, 12, 18, 19, 27, 28, 31, and 38.

Group 1 - Claims 1, 3, 4, 20-22, 32, 34, and 40

Appellants discuss claim 1 as representative of Group 1 claims (Br8-9). We agree with this grouping since claim 1 is the broadest claim in the group. We also briefly touch on some limitations of independent claims 20 and 32, although the claims in this group stand or fall together with claim 1.

Initially, we note that claim 1 does not require doing anything with the generated profile data and does not require an optimizing compiler; compare claim 13. The profile data could be used by a human to perform optimization as taught by Profiler. Thus, Aho is not necessary to the rejection of claim 1.

Turbo Profiler is a program for profiling a program, i.e., for collecting statistics about the run-time operation of the program. It was well known in the computer art (spec. at 9, line 15, to page 10, line 3), that one or more "source modules" are compiled into "object modules" and then linked together to form a single executable program module (the "program"), as recited in the preamble of claim 32. Profiler indicates that a program can be compiled from several modules (e.g., p. 46: "When you choose View|Module, a list box appears that lists all the source modules linked with the program currently loaded into the

Module window."; p. 110: "If your source consists of 10,000 lines in ten modules, you should probably analyze only one module at a time in active analysis."; p. 115: "In very large programs, limit your selection of area markers to a single module per profile run."). However, claim 1 does not require the "executable program module" to be created from more than one source module. Each program (executable program module) in Profiler is composed of one or more "routines," where a "routine" refers in a generic way to functions and procedures (p. 5), and corresponds to the claimed "procedure."

The user of Profiler determines what parts or "areas" of the program to profile. Profiler states (p. 109):

> An *area* is a location in your program where you want to collect statistics: It can be a single line, a construct such as a loop, or an entire routine. An *area marker* sets an internal breakpoint. Whenever the profiler encounters one of these breakpoints, it executes a certain set of code--depending on the options you've set for the area in question. This profiling could be a bookkeeping routine or a simple command to stop program execution.

The areas are set using the Add Areas menu (p. 50). The "area markers" and the associated bookkeeping code in Profiler correspond to the instrumentation code "hooks" described in connection with prior art profilers (spec. at 8, lines 10-13; spec. at 9, lines 1-6). Thus, the program with the area markers inserted in Profiler is "an instrumented executable program module," as recited in claim 1. The program is inherently stored on "storage media," which is broad enough to read on both the computer memory or storage such as a hard disk, because it must be stored somewhere in order to be utilized by the computer.

An example of profiling is described in Chapter 4. The source module PTOLL.C (pp. 129-130) contains three routines: "main()," "route66()," and "highway80()." If the area is set to "Routines in Module," which "adds area markers for all routines in the current module" (p. 50), Profiler will collect information about the time spent in each routine (procedure) and the number of execution counts each time the routine is executed, as shown in Figure 4.1 (p. 130). Each routine is "an instrumented code block," as claimed. (Other possible examples of "instrumented code blocks" are a single line of code or a programming construct such as a loop (pp. 12, 109)). The time-collection compartments and the count-collection compartments in Figure 4.1 are "at least one procedure specific data storage area" because they correspond to data storage areas for time and count data that is specific to each routine (procedure). Appellants give a similar example of three procedures with counter areas (spec. at 18):

> Therefore, if source module 1 had three procedures "main," "foo" and "bar," its corresponding profile data file will have three procedure counter areas identified as "main," "foo" and "bar." The profile data files may be stored or archived with their corresponding source modules for later retrieval.

During execution of the profiled program in Profiler, routine specific data is stored on stacks (p. 131) and after profiling is completed, the "[Statistics|Save] command saves the statistics to a .TFS (Turbo Profiler Statistics) file" (p. 113), which is a profile data file having time and counter storage areas for each area which was monitored. The arrangement or data structure of the "procedure specific data storage area" is not claimed and therefore does not distinguish over the storage in Profiler.

Profiler keeps track of statistics on each module of the program and each area in each module and stores area specific data in a .TFS file, where "[a]n area can be a single line, a construct such as a loop, or an entire routine" (p. 12). Profiler teaches that the areas to be profiled can be set using the Add Areas menu for all routines (procedures) in all modules of the program (p. 50): "Modules with Source adds area markers for all routines in modules whose source code is available." The collected statistics for the program can be viewed in the Execution Profile window (pp. 13-14 & 55-56) by using the All choice from the Filter command (p. 67) or the user can choose only statistics for one module using the Module choice from the Filter command (p. 68). In the examples of Figure 1.2 (p. 13), Figure 1.4 (p. 15), and Figure 1.5 (p. 16), each line in the Execution Profile window has four fields (pp. 13-14): (1) an area name comprised of a module name (PRIME0) and an area name (31 for line 31 in Figure 1.2); (2) the number of seconds spent in that area (6.2655 sec. for line 31 in Figure 1.2) or the number of times that line executed (15,122 times for line 22 in Figure 1.4) or both (Figure 1.5); (3) the percentage of total execution time spent in that area (93% for line 31 in Figure 1.2) or the percentage of total counts (82% in Figure 1.4) or both (Figure 1.5); and (4) a magnitude bar displaying a proportional graph of time (Figure 1.2) or counts (Figure 1.4) or both (Figure 1.5). Thus, Profiler stores profile information about each module and each area (routine or procedure) in the module.

Therefore, except for the need to explain the different terminology, we find the subject matter of claim 1 to be anticipated by Profiler. In fact, since no specific structure is recited for the "procedure specific data storage areas," claim 1 is so broad that it is anticipated by the admitted prior art of "instrumenting profilers" (spec. at 8-9) since known prior art instrumenting profilers must save the count statistics for each block of code or path in storage somewhere.

Appellants' arguments are based on reading limitations from the specification into the limitation of a "procedure specific data storage area." Appellants argue (Br8-9):

> Claim 1 ... requires that profile information for a procedure be stored in a procedure specific data storage area for that procedure. Utilizing such a hierarchical organizational structure facilitates future optimization by permitting the procedure specific data storage area

> associated with the procedure to be analyzed during optimization to determine whether profile data for that procedure exists and is valid. As described above, this permits a computer program to be optimized after modifications made thereto subsequent to profiling, and as a consequence, much of the time and expense that would otherwise be associated with re-profiling a computer program may be eliminated.

The examiner responds that "the limitation '***hierarchical organizational structure facilitates***' is being read into the limitations of Claim 1[] from the Specification and dependent claims" (EA23).

We agree with the examiner that claim 1 does not recite a "hierarchical organizational structure." The specification describes "storage areas, referred to as module counter areas (MCA's [sic, MCAs]) and procedure counter areas (PCA's [sic, PCAs])" (spec. at 17, lines 17-19). Each module has its own profile data file (MCA) (elements 30 in Fig. 1; MCA data structure shown in Fig. 2), which includes one or more procedure counter areas (PCAs) (Fig. 2; PCA data structure shown in Fig. 3; spec. at 18, lines 5-9; spec. at 19, line 19 to page 20, line 16). However, these data structures for the MCAs and the PCAs are not claimed in claim 1. Since appellants have not used the exact term "procedure counter area" from the specification, we will not interpret "procedure specific data storage area" to include all of the disclosed limitations of a PCA. A "procedure specific data storage area" is broad enough to read on any storage area that stores profile data related to a procedure or routine, which is taught by Profiler.

Claim 20 recites "a hierarchical profile data storage system ... including a mechanism for creating unique module counter areas for each program module and unique procedure counter areas for each procedure." Claim 32 recites "for each source code module, initializing a module counter area; for each procedure in the module, initializing a procedure counter area within said module counter area." The claims in Group 1 are argued by appellants to stand or fall with claim 1 and, so, these other claim limitations are not at issue. However, we offer some comments on claims 20 and 32 for appellants' benefit. Profiler stores profile information about each module in the program and each area (routine or procedure) in the module. This data has a "hierarchical" relationship as broadly recited in claim 20; the data structure is not specifically claimed. Profiler must have storage areas corresponding to module counter areas and procedure counter areas as recited in claim 32 in order to be able to display the statistics according to the module and area (procedure) in the Execution Profile window (pp. 13-14 & 55-56). The "module storage area" in claim 32 consists only of "procedure counter areas" and the storage area for the count data for procedures for a module in Profiler is considered a "module storage area." Although claim 32 uses the terms "module counter area" and "procedure counter area," claim 32 is not separately argued for Group 1.  Furthermore, appellants have not argued that all the data structures of Figs. 2 and 3 should be read into these terms; e.g., that the procedure counter area in Fig. 3 should be interpreted in light of the specification as having a header information 50, control flow counters 52, direct call site counters 54, and indirect call site counters 56.

Appellants further argue (Br9):

> Applicants' claimed utilization of procedure specific data storage areas is in contrast to conventional profilers that store and organize profile data merely on a <u>program-by-program</u> basis.  The profile data is generated for various regions of a program based upon the insertion of profiling hooks within specific regions of a program.  Certainly, if a hook is placed in a specific procedure, profile data specific to that procedure will be created.  However, there is no disclosure or suggestion in the art of <u>storing</u> such profile data in a procedure specific data storage area.  Storage of the profile data in conventional profiles is still on a program-by-program basis.

The examiner responds that procedure storage areas are taught by Profiler (EA24).

Profiler stores profile information about each module of the program and each area (routine or procedure) in the module as evidenced by the fact that it can display the profile statistics according to the module and area (procedure) in the Execution Profile window (pp. 13-14 & 55-56).  The procedure counter and time areas shown in Figure 4.1 (p. 130) clearly show that Profiler stores procedure specific data.  Claim 1 does not preclude the "procedure specific data storage area" from being part of a larger storage area, such as the .TFS file in Profiler that stores all program profile information.

Appellants further argue (Br9):

> The examiner relies principally on *Profiler* for allegedly disclosing the use of procedure specific data storage areas. However, it appears that the Examiner is confusing the concept of <u>creating</u> profile data with the concept of <u>storing</u> profile data, the latter of which is the focus of claim 1. Indeed, the Examiner's response to Applicants' arguments made at pages 6 and 7 of the Office Action dated December 17, 1999 focus on the general organization of a computer program into modules and procedures, and specifically only on the <u>creation</u> of profile data. Applicants are not claiming as novel the concept of creating profile data that is specific to a particular procedure. Rather, it is the unique <u>organizational structure</u> within which such data is stored that is distinguishable from the prior art of record.

The relevant argument is in the last sentence. The argument that "it is the unique <u>organizational structure</u> within which such data is stored that is distinguishable from the prior art of record" (Br9) is not commensurate in scope with claim 1 because the "procedure specific data storage area" has not been defined to have any particular organizational structure. Any data storage area that stores data related to statistics of a procedure (routine) is "procedure specific data storage area." Profiler teaches storing area specific data in a .TFS file, where "[a]n area can be a single line, a construct such as a loop, or an entire routine" (p. 12). As to independent claims 20 and 32, which are not argued with respect to Group 1, these claims do not recite any organizational structure which defines over Profiler. Profiler stores profile information about each module of the program and each area (routine or procedure) in the module.

Appellants further argue (Br9-10):

> *Profiler*, in particular, is focused only on the creation of profile data, and not how that data is logically arranged in storage. As shown, for example, at pages 90-92 of *Profiler*, profile statistics for a program are stored and retrieved in .TFS files, each of which organizes profile data on a program-by-program basis. Moreover, as shown at pages 47 and 48 of *Profiler*, the concept of a "module window" is discussed, explaining how source modules in a program may be loaded into a module window. Nonetheless, even though *Profiler* discusses the possibility of parsing a program into multiple modules, profile data storage is still performed in program-wide files. *Profiler* recognizes the concept of a module, yet does not disclose or suggest any mechanism for collecting and storing such information even on a module-by-module basis, much less on a procedure-by-procedure basis, as is specifically recited in claim 1.

These arguments are not persuasive. Claim 1 does not recite how procedure specific data is logically arranged in storage. Profiler stores profile information about each module of the program and each area (routine or procedure) in the module as evidenced by the fact that it can display the profile statistics according to the module and area (procedure) in the Execution Profile window (pp. 13-14 & 55-56). The procedure counter and time areas shown in Figure 4.1 (p. 130) clearly show that Profiler stores procedure specific data.

For the reasons discussed above, appellants have not shown any error in the rejection of claim 1 in Group 1. The rejection of claims 1, 3, 4, 20-22, 32, 34, and 40 is sustained.

Group 2 - Claim 33

The examiner points, without explanation, to Profiler, pages 130-132 (Paper No. 7, p. 14). These pages cover a section entitled "Who pays for loops?" It is not explained, nor do we understand, how this section is intended to be applied against the limitations of claim 33.

Appellants argue that Profiler discloses storage of profile data on a program-by-program basis and therefore does not suggest the use of module specific files (Br11).

The examiner responds that Profiler captures the same information, but uses a different data structure (EA31). The examiner states that "[d]ata structures are not patentable and the Examiner holds the functionality equivalent" (EA31).

We agree with the examiner that Profiler captures the same information as claimed: count information for each procedure within each module. However, while data structures per se are non-statutory subject matter, see In re Warmerdam, 33 F.3d 1354, 1361-62, 31 USPQ2d 1754, 1760 (Fed. Cir. 1994), it is not true that data structure limitations in a claim to a product can be disregarded, see In re Lowry, 32 F.3d 1579, 1582, 32 USPQ2d 1031, 1034 (Fed. Cir. 1994). The so-called "point of novelty" approach where the non-statutory subject matter in a claim (e.g., a mathematical algorithm _per se_) is ignored has been consistently rejected, _see_ _Diamond v. Diehr_, 450 U.S. 175, 188-89, 209 USPQ 1, 9 (1981) (claims must be considered as a whole), with the possible exception of printed matter where it bears no functional relationship to the substrate on which it is printed, _see_ _In re Gulack_, 703 F.2d 1381, 1386, 217 USPQ 401, 404 (Fed. Cir. 1983). Further, it was improper for the examiner to dismiss the differences by just stating that the data structures are functionally equivalent. The issue is not equivalence, but obviousness. _See_ _In re Edge_, 359 F.2d 896, 898, 149 USPQ 556, 557 (CCPA 1966); _In re Ruff_, 256 F.2d 590, 599, 118 USPQ 340, 348 (CCPA 1958) (the equivalence must be disclosed in the prior art or be obvious within the terms of § 103). Nevertheless, we conclude that the subject matter of claim 33 would have been obvious to one of ordinary skill in the computer programming art.

Initially, we clarify the issue. Claim 33 recites: "The method of claim 32 further comprising the steps of: creating a profile file for each module counter area; and including within each profile file said count information collected for each relevant procedure counter area." A "file" is defined in computer science as "a collection of bytes stored as an individual entity." Thus, claim 33 requires that each module counter area (MCA) is a separate profile data file as shown in

Fig. 1. Independent claim 32 does not recite how the module counter area for each module and the procedure counter area for each procedure in the module are stored; the profile storage could be a single file which contains all procedure counters for one or all modules as taught by Profiler. Profiler stores profile information about each module of the program and each area (routine or procedure) in the module as evidenced by the fact that the collected statistics for the program can be viewed in the Execution Profile window by module name and area name (pp. 13-14 & 55-56). The storage area associated with procedures for a given module can be considered a module counter area since the module counter area is not recited to consist of anything other than procedure counter areas. Profiler discloses that the profile data statistics program are stored in a .TFS file. Thus, the arguable difference between Profiler and the subject matter of claim 33 is that Profiler does not expressly teach storing procedure counter data for each module in a separate profile file. Again, no optimizing compiler is claimed, so Aho adds nothing to the rejection.

Profiler discloses that for large programs involving several modules, profiling should be done on one module at a time (e.g., p. 110: "If your source consists of 10,000 lines in ten modules, you should probably analyze only one module at a time in active analysis."; p. 115: "In very large programs, limit your selection of area markers to a single module per profile run."). One of ordinary skill in the computer art would have known that the profile data for one module would be saved as a single file corresponding to the claimed "profile file." Thus, it would have been obvious to one of ordinary skill in the art that profiling data for procedures and modules in Profiler could be saved either as one file, containing all procedure data for all modules, or as one file per module. The rejection of claim 33 is sustained.

<u>Group 3 - Claims 5-7, 13-17, 23, 24, 29, 30, 35, 36, and 39</u>

The independent claims in this group require an optimization mechanism that (1) determines if procedure specific profile information exists for a procedure, and (2) determines if the existing profile information is valid. As described in the specification (spec. at 14, lines 15-18): "[P]rofile data will be said to be 'valid' either if the corresponding procedure has not changed, or if the data is considered sufficiently adequate (e.g., it is similar enough to the original procedure) and the compiler can still use the data in this fashion."

Appellants argue that neither Profiler nor Aho discloses or suggests an optimizer that checks for each procedure in a module to determine whether both existing and valid profile information is present (Br12). It is argued that Aho discusses optimization but presumes that all necessary profile information is available and can be used during optimization (Br12). It is argued that by checking for both the existence and validity of profile information for each procedure during optimization, the compiler is able to perform optimization in many instances without requiring profiling to be repeated on a modified program (Br12).

The examiner finds that Aho teaches several forms of commonly used data validation, such as "type checking," and that validation of data is routine in programming (EA34). The examiner concludes that the broadest reasonable interpretation covers the recited claim limitations (EA34).

While we agree with the examiner that data validation is routine in compiling, this does not make all validation obvious. This following analysis only addresses the claim limitations and gives no weight to the arguments about the way the validity data may be used because this is not claimed. Profiler generates profile information and discusses various optimization techniques to be performed by a human (e.g., pp. 113-116; 124-128). Profiler does not perform any optimization by itself and, thus, has no need to determine whether profile information is valid. Aho discloses that compilers which perform code-improving transformations are called optimizing compilers (p. 585). Aho discloses (p. 585): "Profiling the run-time execution of a program on representative input data accurately identifies the heavily traveled regions of a program. Unfortunately, a compiler does not have the benefit of a sample input data, so it must make its best guess as to where the program hot spots are."  As indicated in the second sentence, the code optimizations described in Chapter 10 of Aho are not based on profile information because this is generally not available.  However, Aho reasonably suggests to one of ordinary skill in the compiler art to concentrate the optimization techniques on heavily traveled regions of a program as determined by a profiler.  In any case, it is admitted that optimizing compilers which use profile information were known in the computer art (spec. at 11, lines 8-10: "Compilers can also automatically read in profile information during an optimization phase to create an optimized version of the computer program.").  However, none of Profiler, Aho, or the admitted prior art discloses or suggests determining if existing profile information is valid.  The fact that compilers may determine if other information is valid does not suggest the obviousness of this limitation.  We conclude that the examiner has failed to establish a <u>prima facie</u> case of obviousness with respect to the claims of Group 3.  The rejection of claims 5-7, 13-17, 23, 24, 29, 30, 35, 36, and 39 is reversed.

<u>Group 4 - Claims 8-10, 25, 26, and 37</u>

The claims in this group all depend on claims in Group 3 and define how the optimization mechanism determines validity.  For example, representative claim 8 recites that validity is determined "by comparing a signature of each procedure with information stored in each corresponding procedure counter area." A "signature" is described in the specification (spec. at 23, lines 17-22). Claims 25 and 26 do not recite "signatures," but determine validity by the same kind of comparisons.

Since neither Profiler nor Aho teaches or suggests determining validity, as discussed in the analysis of Group 3, they do not teach or suggest the specific mechanisms for determining validity in the claims of Group 4. Accordingly, the rejection of claims 8-10, 25, 26, and 37 is reversed.

Group 5 - Claims 11, 12, 18, 19, 27, 28, 31, and 38

The claims in this group recite that the optimization mechanism additionally includes a mechanism that constructs a call graph from profile data and a mechanism that analyzes the call graph to determine a procedure packaging order which omits procedures that no longer exist. Appellants argue that this allows optimization of a packaging order despite the absence of some profiling data (Br15). It is further argued that "[n]either *Profiler* nor *Aho* discloses or suggests the performance of optimization when only partial profile data is available" (Br15).

The examiner finds that appellants fail to address the kind of information gathered for optimization in Profiler at Table 3.1

(p. 114) and the fact that call graphs are inherent in compilers (EA38). The examiner observes that Aho teaches optimization based on collected performance data (EA38).

We do not consider the examiner's reasoning persuasive of obviousness because it does not address the specifics of the claimed subject matter. The fact that call graphs and optimizing compilers were known, as evidenced by Chapter 10 of Aho, does not address the specific limitations of a call graph based on profile data or determining a procedure packaging order as claimed. Neither Profiler nor Aho discloses constructing a call graph based on profile data or determining a procedure packaging order. The specification is more relevant than any art cited by the examiner and admits that "[k]nown in the art are existing methods that analyze a weighted call graph of an object module or executable module and rearrange the procedures in that module to improve spatial locality, thus making more efficient use of memory paging systems" (spec. at 24, lines 7-10). However, the specification says nothing about a mechanism that "omits procedures that no longer exist" (claim 8). While Aho discloses dead-code elimination (p. 595), this is not in connection with packaging and the examiner does not rely on this teaching of Aho or on the admitted prior art. The optimization described in Aho is not concerned with optimizing spatial locality by determining a procedure packaging order. We conclude that the examiner has failed to establish a prima facie case of obviousness with respect to the claims of Group 5. The rejection of claims 11, 12, 18, 19, 27, 28, 31, and 38 is reversed.

## CONCLUSION

The rejection of claim 26 under 35 U.S.C. § 112, second paragraph, is reversed.

The rejection of claims 1, 3, 4, 20-22, 32-34, and 40 under § 103(a) is sustained. The rejection of claims 5-19, 23-31, and 35-39 under § 103(a) is reversed.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 CFR § 1.136(a).

AFFIRMED-IN-PART

```
JERRY SMITH                      )
Administrative Patent Judge      )
                                 )
                                 )
                                 )
                                 )  BOARD OF PATENT
LEE E. BARRETT                   )     APPEALS
Administrative Patent Judge      )       AND
                                 )   INTERFERENCES
                                 )
                                 )
                                 )
MAHSHID D. SAADAT                )
```

Administrative Patent Judge )

Scott A. Stinebruner
WOOD, HERRON & EVANS LLP
2700 Carew Tower
441 Vine Street
Cincinnati, OH  45202-2917